6

Objektno orijentisano programiranje
vežbe

Staša Vujičić Stanković

ZADACI :

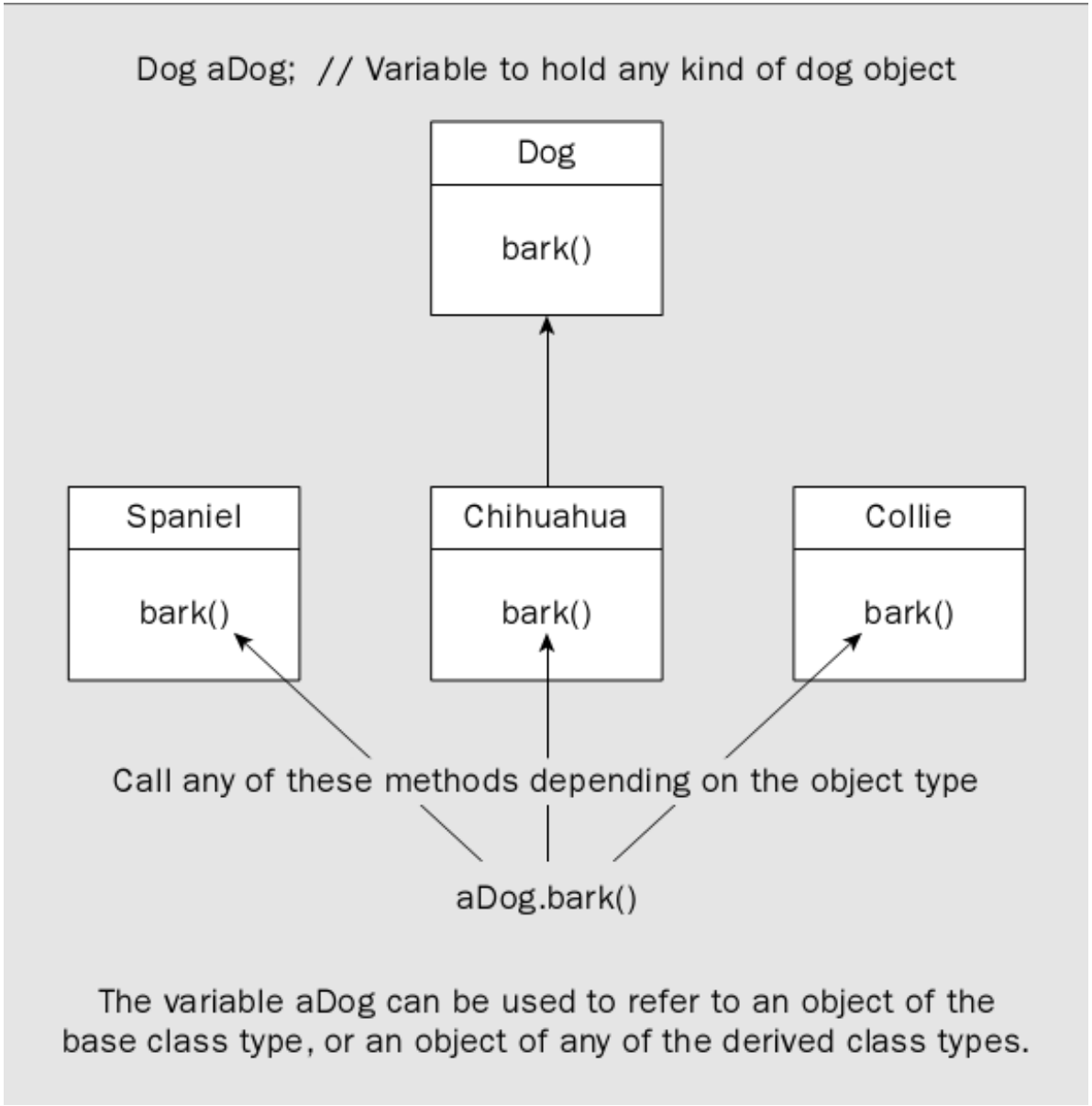
- Prilikom čitanja slajdova
  OBAVEZNO uporedo čitati zadatke iz foldera
  polimorfizam.

Polimorfizam

- Polimorfizam označava mogućnost
  da se jedan isti poziv metoda
  ponaša različito u zavisnosti
  od tipa objekta na koji se metod primenjuje.

- Polimorfizam radi sa objektima izvedene klase!

POLIMORFIZAM



Polimorfizam

- Koji će metod biti pozvan,

  zavisi od tipa objekta nad kojim se metod poziva,

  a ne od tipa promenljive koja sadrži referencu na objekat!

- **Napomena!**

  Polimorfizam se primenjuje isključivo na metode,
  ne i na članice podatke.

Polimorfizam

- Za polimorfizam je neophodno sledeće:
  - Poziv metoda nad objektom izvedene klase vrši se preko promenljive bazne klase.
  - Metod koji se poziva mora biti deklarisan u baznoj klasi.
  - Metod koji se poziva mora biti definisan u izvedenoj klasi.

- Potpis metoda u baznoj i izvedenoj klasi mora biti isti.

- Povratni tip metoda u izvedenoj klasi mora biti isti kao i u baznoj klasi ili tipa koji je potklasa klasnog tipa bazne klase (tada kažemo da su povratni tipovi kovarijantni).

Primer

```
public class Zivotinja {
        Zivotinja vratiObjekat(){ … }
   }

public class Pas extends Zivotinja {
        Pas vratiObjekat(){ … }
   }
```

Apstraktne klase

- **Apstraktna klasa** je klasa koja ima jedan ili više metoda koji su deklarisani, a nisu definisani – nazivamo ih **apstraktnim metodima**.

- Deklaraciji apstraktnog metoda prethodi ključna reč **abstract** i oni nemaju telo metoda.

Apstraktne klase

- U prvom redu definicije klase
  navodi se ključna reč abstract,
  da bi se klasa deklarisala kao apstraktna.

- Apstraktni metod ne može biti private,
  jer private metod ne može biti nasleđen,
  a samim tim ni redefinisan u potklasi.

Primer

```
public abstract class Zivotinja {
        public abstract void zvuk();
}
```

- Ovim smo klasu Zivotinja deklarisali kao apstraktnu, pošto sadrži apstraktni metod zvuk().
- Metod zvuk() će biti predefinisan u izvedenim klasama.

Apstraktne klase

- Ne može se instancirati objekat apstraktne klase, ali je moguće deklarisati promenljivu tipa apstraktne klase.

  Zivotinja ljubimac = null;

- Ova promenljiva se kasnije može koristiti da sačuva referencu na konkretan objekat izvedene klase.

Apstraktne klase

- Ukoliko nisu svi apstraktni metodi bazne klase definisani u izvedenoj klasi,
  ona će takođe biti apstraktna klasa.

- Ako je klasa apstraktna,
  mora se koristiti ključna reč abstract
  prilikom njene definicije.

Univerzalna natklasa

- Sve klase koje definišemo su podrazumevano potklase klase Object iz paketa java.lang.

- Posledice:
  - Promenljiva tipa Object može da čuva referencu na objekat proizvoljnog tipa.
    Ovo je korisno kod metoda koji mogu da prihvate kao argument objekte različitog tipa.
  - Klasa koju definišemo nasleđuje članice klase Object. Sve članice klase Object su metodi.

Klasa Object

- Bitniji public metodi klase Object:
  - **toString()** – vraća String reprezentaciju objekta.
    Ukoliko u klasi ne predefinišemo metod toString()
    on vraća:
    ime_klase@heksadekadna_reprezentacija_objekta.
  - **equals()** – poredi referencu na objekat
    prosleđen kao argument i referencu na tekući objekat
    i vraća true ukoliko su jednake, tj. referišu na isti objekat.
    U suprotnom, vraća false,
    čak i kad objekti imaju jednake vrednosti svojih članica.
  - **getClass()** – vraća objekat tipa Class koji identifikuje
    klasu tekućeg objekta.
    Ne može se preklapati, deklarisan je sa final.

Klasa Object

- Protected metodi klase Object:
  - **finalize()** – automatski se poziva pre nego što je objekat konačno uništen i memorija oslobođena. Može se predefinisati u klasi – na primer koristan je ako objekti klase koriste resurse koji traže neku specijalnu akciju kada bivaju uništeni, kao što je slučaj sa grafičkim resursima, fontovima i sl. Metod finalize() takođe daje informaciju da je objekat zaista uništen.

ODREĐIVANJE TIPA OBJEKTA

- Class tipObjekta = ljubimac.**getClass()**;
  /* Promenljiva *ljubimac* je tipa *Zivotinja* i referiše na konkretan primerak klase *Zivotinja*. */

- System.out.println(tipObjekta.**getName()**);
  /* metod getName() je član klase Class i vraća puno kvalifikovano ime klase */

- **Napomena!** Objekat tipa Class se odnosi na aktuelnu klasu nekog objekta.

```
String s = "Ovo je string";
Object str = s;

System.out.println(str.getClass().getName());
        /* rezultat je  java.lang.String */
```

Kastovanje objekata

- Možemo vršiti kastovanje objekata u drugi klasni tip, ali samo ukoliko su ove klase u istoj hijerarhiji izvedenih klasa.

```
Spaniel a = new Spaniel("Endi");
Zivotinja a1 = (Zivotinja) a;
```

KASTOVANJE OBJEKATA

- Kada se radi kastovanje u potklasni tip,
  to mora eksplicitno da se navede,
  pri čemu objekat mora biti legitimna instanca klase u koju vršimo kastovanje.

  Pas a2 = (Pas)a1;

- Razlozi za kastovanje:
  - Kada izvršavamo metode polimorfno
  - Da prenesemo objekte nekoliko mogućih klasa metodu

KASTOVANJE OBJEKATA

Patka patka = new Patka(“Donald", “Pekinska");

Zivotinja ljubimac = patka;

ljubimac.lezeJaja();

// Promenljiva ljubimac referiše na objekat

// klase *Patka*, ali i pored toga ova naredba

// dovodi do greške. Razlog je što metod

// *lezeJaja()* ne postoji u klasi *Zivotinja*.

Ispravno je napisati sledeće:

((Patka)ljubimac).lezeJaja();

Identifikacija objekata

- Operator *instanceof*

  if(kucniLjubimac instanceof Pas) ...

- Proverava da li je moguće izvršiti kastovanje objekta referisanog levim operandom u tip sa desne strane,
  tj. rezultat je **true** ako je objekat istog tipa kao i desni operand ili bilo kog od podtipova.